Page 10, please amend the paragraph beginning at line 9, as follows:

In a second implementation of the invention, a data object to be archived ~~comprises~~ may comprise one or more fields of one or more tables, and the ID of the respective object ~~comprises~~ may comprise one or more key fields of that data object. This can best be seen from Fig. 2. In this instance, various sets of data objects are created in the form of two-dimensional data arrays, i.e., two tables ~~201, 202~~ (table 1 and table 2) having columns named field A to field X and field A to field Y, respectively, and a certain, unspecified number of lines. A field of the array or table is defined by the name of the column and the respective line. Such a field can contain data to be archived. It can alternatively contain a reference to a line of a further table. For example, in table 1 field X in line 2 contains a reference to line 3 in table 2. A data object 201.x to be archived comprises of fields of one line of the respective table. If one of the fields contains a reference to a line of an other table, fields of this referenced line belong to the data object, as well ~~too~~. In the example in Fig. 2, a data object 201.x to be archived comprises the fields of line 2 in table 1 and fields of line 3 in table 2.

Page 10, please amend the paragraph beginning at line ~~31~~ 30, as follows:

An ID of such a data object can be implemented by the content of one or more so-called key fields, if the combination of these key fields is unique within the respective table. In the example, the fields of "field A" and "field B" can be used as key fields for table 1, whereas field A alone is key field in table 2. Within this example, assigning an ID to the data object means to use the content of the fields of columns field A and B of the respective lines as the ID for that particular line. In line with this assignment, the IDs

for the data object 201.x to be archived are stored as a first type ID in a first type lock object 203 named permanent lock object in Fig. 2 and as a second type ID in a second type lock object 204 named transactional lock object. The permanent lock object ~~is~~ may be implemented as a table having two columns, the first of which contains the first type ID 1. The second type ID, ID 2, can be implemented as a one-dimensional data array stored in the working memory of the computer system. However, it can be implemented as a file on a nonvolatile storage means, too. The first type ID, ID 1, is deleted after the selected data object has been deleted according to ~~step g) of~~ the inventive process, and the second type ID, ID 2, ~~is~~ may be deleted immediately after the time as defined in step h). Alternatively, type ID 1 IDs can be deleted after all the selected data objects have been deleted according to step g). As can be seen, both ID types have identical content, the ID of the respective lines of the data to be archived. However, this is not a necessary condition. Different contents can be used for the different ID types. The permanent lock objects further contain a column by which a filename ~~is~~ may be assigned to the ID of the data object, i.e. that data object to be archived. In the example, line 1 is archived in a file named 001, lines 2 and 3 in file 002, and line 4 in file 003.

Page 12, please amend the paragraph beginning at line 7, as follows:

A further embodiment ~~is characterized in that in step e) the second type ID is stored~~ may comprise storing the second type ID in the second lock object immediately after ~~performing step e)~~ assigning an ID of a second type to each of the selected data objects for the respective data object. Alternatively, ~~in step e)~~ the second type of ID of

the selected data object is stored ~~shortly~~ before ~~the~~ storing ~~process according to step f) for~~ the data object assigned to that ID is started.

07115108

Page 12, please amend the paragraph beginning at line ~~15~~ 14, as follows:

A further embodiment ~~is characterized in that in step d)~~ may comprising storing the first type IDs of all selected data objects ~~are stored~~ before ~~the first~~ storing the data object at the second storage location ~~process according to step f) is started~~.

07115108

Page 12, please amend the paragraph beginning at line ~~20~~ 19, as follows:

In a further embodiment the invention ~~comprises j)~~ may comprise checking ~~before or while performing any of steps a) to e) for a data object,~~ whether a first type ID for the data object has been stored in a lock object, and if ~~yes~~ the first type ID has been stored, ~~skipping at least step f)~~ not storing the data object, the first ID of which is contained in the first lock object, at the second storage location for that data object.

07115108

Page 12, please amend the paragraph beginning at line ~~26~~ 25, as follows:

Additionally, the invention ~~comprises k)~~ may comprise checking ~~before or while performing any of steps a) to f) for a data object,~~ whether that data object is contained in the second storage location, and if ~~yes~~ the data object is contained, ~~skipping at least step f)~~ not storing the data object, the first ID of which is contained in the first lock object, at the second storage location for that data object.

07115108

Page 12, please amend the paragraph beginning at line ~~32~~ 31, as follows:

Another embodiment ~~is characterized by said~~ may comprise checking ~~is performed~~ by querying a first lock object.

07115108

Page 12, please amend the paragraph beginning at line 34, as follows:

Still another embodiment ~~comprises, l) in case of a failure in step f)~~ may comprise determining whether the data object, the first ID of which is contained in the first lock object, was stored successfully at the second storage location, and upon a successful storage, checking, whether the data object assigned to the respective first ID has been completely stored in the second storage location, and ~~in case of no~~ if the respective ID has not been completely stored, skipping ~~at least steps g) and h)~~ the step of deleting the data object from the first storage location and the step of deleting the first type ID from the first lock object after the respective data object assigned to that at least one ID has been for that data object and deleting the first ID from the first lock object.

07115108

Page 13, please amend the paragraph beginning at line 6, as follows:

Embodiments of [[The]] the invention [[is]] are now described in more detail with reference to Figs. 3 to 5, which are schematic flow diagrams of exemplary methods may be implemented by ~~implementations of~~ the selecting, writing and deleting modules, respectively, as shown in Fig. 1. Within the context of this description, and particularly ~~the~~ with respect to Figs. 3 to 9, a first type ID is called a P-lock (permanent) and a second type ID is called a T-lock (transactional). ~~So~~ Therefore, setting a P- or T-lock for a selected object means to store an ID of that object in a respective lock object. The term "permanent" results from the property of the P-lock of existing permanently, as long as the data object is not yet deleted from its original storage location. The term "transactional" results from the property of the T-lock of existing only as long as a specific action (e.g., checking of archiveability) is performed on a selected data object

or, in other words, of being deleted ~~shortly~~ after the respective action has been performed.

07115108

Page 13, please amend the paragraph beginning at line ~~29~~ 26, as follows:

In the exemplary flow chart of the selecting module in Fig. 3, a data object is selected in a first step 301. Subsequently, a T-lock is set on this object in step 302. If the T-lock was successfully set (step 303), that is, if it did not yet exist, it is checked in step 304 whether a P-lock already exists in the selected data object. If the T-Lock could not be set successfully, the next data object is selected in step 309. The setting of the T-lock (step 302) and the check (step 303), whether it is successfully set, can advantageously be implemented as one "atomic" step. This means that both steps can be executed essentially at the same time or, in other words, the time gap between both steps can be essentially zero.

07115108

Page 14, please amend the paragraph beginning at line ~~9~~ 6, as follows:

Both checks (steps 303 and 304) can also be implemented by querying the respective lock objects. If a P-lock exists, the T-lock is deleted (step 308) and the next data object is selected (step 309). If no P-lock exists, it is checked in steps 305 and 306[[,]] whether the data object is archiveable. Such checking comprises a test whether the data in the data object is readable, complete[[,]] not being fraught with obvious failures etc. If the test was successful, a P-lock is set on that data object in step 307, whereby no archive file is assigned to the data object. Then the T-lock is deleted (step 308) and the next data object can be selected (step 309).

07115108

Page 14, please amend the paragraph beginning at line 23, as follows:

In the exemplary flow chart of the writing module in Fig. 4, a data object is selected in a first step 401. Subsequently, a T-lock is set on this object in (step 402). If the T-lock was successfully set (step 403), it is checked in step 404 whether a P-lock already exists in the selected data object, whereby no file must be assigned to that data object. If the condition is not fulfilled, the T-lock is deleted in step 407, and the next data object is selected in step 408. If a P-lock exists, the data object is stored in an archive file in step 405 and the archive file is assigned to the data object in step 406, e.g. by adding the file name to the lock object as shown in Fig. 2. Subsequently, the T-lock is deleted (step 407), and the next data object is selected (step 408).

07115108

Page 15, please amend the paragraph beginning at line 4, as follows:

In the exemplary flow chart of the deleting module in Fig. 5, a data object, that has already been archived is selected (step 501). This can be implemented by checking the archive files. If a data object has been selected and successfully read from the archive file, that data object is deleted from the original storage location (step 502), the P-lock is deleted (step 503), and the next data object is selected (step 504).

07115108

Page 15, please amend the paragraph beginning at line 18, as follows:

In the exemplary flow chart of a further exemplary implementation in Fig. 6, the selecting and writing module described above are combined to one module. Accordingly, a data object is selected in a first step 601. Subsequently, a T-lock is set on this object in step 602. If the T-lock was successfully set step 603, it is checked in step 604 whether a P-lock already exists in the selected data object. If the T-lock could

not be set successfully, the next data object is selected (step 611). If a P-lock exists on that object, the T-lock is deleted in (step 610) and the next data object is selected (step 611). If no P-lock exists on that object, it is checked in (step 605)[[,]] whether the data object is archiveable. If this check fails (step 606), the T-lock is deleted (step 610) and the next data object is selected (step 611). If the check 606 is positive, a P-lock is set (step 607), the data object is stored (step 608) in an archive file, the archive file is assigned to the P-lock (609), the T-lock is deleted (step 610), and the next data object is selected (step 611).

Page 15, please amend the paragraph beginning at line 31, as follows:

Fig. 7 shows ~~by way of an exemplary~~ a flow chart of an exemplary method to demonstrate how any software application can use the concept of the P[[-]] and T-locks to ensure that the measures, that the software application is going to apply on the data object, do not influence the archiving process. A software application which is programmed to have a read and/or write access to data objects, which can be subject of an archiving process as described, ~~comprises~~ may comprise the following steps as shown in Fig. 7. In a first step 701, the data object is selected. Then, a T-lock is set in step 702 on that object by the application. If the T-lock is successfully set (step 703), it is checked in step 704, whether a P-lock exists on that object[[,]]; otherwise the application terminates in (step 707). If a P-lock exists on that object, the T-lock is deleted (step 706), and the application terminates (step 707). If no P-lock exists, i.e., the data object is not subject to an archiving process, the application can have read/write

access to the data object in a working step 705. Subsequently, the application deletes the T-lock (step 706) and terminates (step 707).

Page 16, please amend the paragraph beginning at line 18, as follows:

Fig. 8 ~~shows a process alternative to that shown in Fig. 7~~ is a flowchart of another exemplary method to demonstrate how any software application may use the concept of the P and T-locks, including a conditional deletion of a P-lock. In a first step 801, the data object is selected. Then, a T-lock is set on that object by the application (step 802). If the T-lock is successfully set (step 803), it is checked (step 804)[[,]] whether a P-lock exists on that object[[,]]; otherwise the application terminates (step 809). If no P-lock exists (step 804), i.e., the data object is not subject to an archiving process, the application can have read/write access to the data object in working step (step 807). Subsequently, the application deletes (step 808) the T-lock and terminates (step 809). If a P-Lock exists (step 804), it is checked (step 805)[[,]] whether a file is assigned to it. If a file is assigned, the application deletes the T-lock (step 808) and terminates (step 809). If no file is assigned, the P-lock is deleted (step 806), and the application can have read/write access 807 to the data object. Subsequently, the application deletes the T-lock (step 808) and terminates (step 809).

Page 17, please amend the paragraph beginning at line 8, as follows:

Fig. 9 ~~shows an example of~~ is a flow chart ~~for~~ of an exemplary method for implementation by a software module ~~by means of~~ through which the locks set by the modules described above can be deleted. This can be useful in cases in which no archive files are assigned to P-locks or in which P-locks have been deleted for a user.